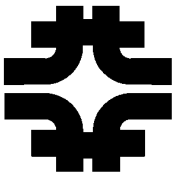
XXXXX



Fermilab

Run II Production Management Architecture

Mark Breitung, Liz Buckley-Geer, Yen-Chu Chen, Phil DeMar, Mike Diesburg, Jim Fromm, Krzysztof Genser, Don Holmgren, Stephan Lammel, Tanya Levshina, Lee Lueking, Igor Mandrichenko, Heidi Schellman, Marilyn Schweitzer, Dane Skow, Steve Wolbers, G. P. Yeh

9/24/98

Version 1.0 - DRAFT

Abstract

Production Management for Run II includes effort and software required for the operation of the off-line production systems. These include the farms, for the event reconstruction, and the systems which are used to split and prepare the data for the final physics analysis. In order to make use of these systems there must be software that schedules and runs “jobs” on the farms and elsewhere, and that ensures that the data is moved to and from storage systems properly.

This document describes the architecture for Run II Production Management and will be used as the basis for subsequent software design documents and hardware configuration analysis.

Fermi National Accelerator Laboratory
Batavia, IL 60510

1.0 INTRODUCTION

The Run II Production Management project includes the software required for the operation of the off-line production systems. These include the farms, for the event reconstruction, and the systems which are used to split and prepare the data for the final physics analysis. In order to make use of these systems there must be software that schedules and runs “jobs” on the farms and elsewhere, and that ensures that the data is moved to and from storage systems properly.

To the user, there are clearly issues of job scheduling, HSM and robotics, calibration constants, accounting and logging, executable version tracking, and other aspects of production systems. All of the systems must be able to smoothly handle the data as it comes from the experiments and also reprocessing of data where required. Other uses of the system for test data, special data sets, Monte Carlo data sets, etc. must also be taken into account.

The scope of the project and its deliverable elements are:

- Proper understanding of the data formats and data flow as part of the production process
- Specification of the input and output of data flow through the Data Access to the Storage Management Software (SMS.)
- Necessary software to efficiently get/pass data to/from computational processes
- Considerations for error detection and recovery
- Necessary database interfaces in order to update and read from the databases for the production work
- Batch system or equivalent for scheduling, controlling, monitoring jobs
- Provide Ability to track job history and system usage.
- Ensure availability of development and test environment
- Appropriate documentation for each of the above elements

This project interfaces with, but does not overlap with, the following other Run II projects:

- Reconstruction Farm and Processing Hardware
  - — Ensure the hardware satisfies our requirements
- Networking Hardware
  - — Ensure the networking satisfies our requirements. Particularly in relation to SMS. For example, a single host may require a dedicated network path to SMS which is separate and non-interfering with interactive access.
- Data Access
  - — Production Management will provide a general purpose batch system as well as a specialized batch system for CPU intensive jobs. The latter will be run on traditional sort of Farm worker host and will not access the SMS directly. The former (among other purposes) may be used for jobs that retrieve/write data from/to SMS.
  - — Production Management will provide software to manage local scratch disk.
  - — Data Access will make requests to retrieve/write data from/to SMS into Farms Scratch disk and subsequently request analysis.
  - — Production Management will not directly access CDF/D0 metadata event databases, but will preclude such access by CDF/D0 jobs.
- Reconstruction Input Pipeline

  - — It is assumed that Data Access is an interface between RIP and Production Management. Should this not be true, Production Management will need to ensure its architecture can directly interface with RIP.
- Physics Analysis Hardware
  - — Some Production Management hosts may overlap with Analysis hosts.
  - — The Production Management batch system (or a separate instance of it) may be used on Analysis systems. It would be convenient for users if the batch API provided for Analysis were similar to the one provided for Production Management.
- Support Databases
  - — Production Management will try to take advantage of the available tools/databases for any of our database needs
  - — Production Management will not preclude access to databases for other CDF/D0 needs
- Software Configuration Management
  - — Production Management must ensure our software fits into this framework
- Storage Management
  - — Production Management will promote and not preclude efficient access to Storage Management, though, most likely, Production Management will not directly access Storage Management through our software.

This document describes several hardware architecture options and the software architecture for Run II Production Management. It will subsequently be used as the basis for subsequent software design documents and hardware configuration analyses.

Throughout this document the term “Farm” or “Farms” will be used interchangeably with the term “Production Management” system.

2.0 REQUIREMENTS

The driving force behind the Farms architecture is to provide sufficient CPU capacity for CDF and D0 reconstruction to keep up with data taking. Extra CPU must also be provided for Monte Carlo, Stripping and Reprocessing. Estimates of the amount of CPU required are described in section 2.1. Because these estimates are so large, it is especially important to maximize the CPU efficiency and keep costs down. Eliminating outmoded and high maintenance technologies and introducing new cost effective technologies is one way to achieve this. Providing a system that may be run efficiently 24 hours a day and 7 days a week helps as well.

A Centralized Mass Storage will have a large role overall in Run II and the Farms must pay particular attention the CDF and D0 data access techniques

Subsequent development and deployment must be completed in time for the following milestones and their estimated dates:

1st Phase of Farms Purchase	Spring, 1999
CDF/D0 Mock Data Challenge	Fall, 1999
First Collisions	April, 2000

2.1 CPU REQUIREMENTS

The Farms must have sufficient capacity to handle the DC rate for event reconstruction. This capacity is based on the following estimated MIPs required for reconstruction

Element		**CDF**		**D0**	
		Min.	**Max.**	**Min.**	**Max.**
Raw Event Size (KBytes)		250		250	
Events/Second	**Peak Hz**	75		50	
	DC Hz	28		20	
MIPs/Event		1200	1800	2000	5000
MIPs to keep up with DC	**100% efficiency**	33,600	50,400	40,000	100,000
		73,600 - 150,400			
	70% efficiency	48,000	72,000	58,000	143,000
		106,000 - 215,000			

where a Fermi MIP is approximately equal to 7 hundredths of a SPECint95 MIP or approximately 3 SPECint92 MIPs.

Considering other types of processing, the number of MIPs required overall are:

Type of Processing		Estimated MIPs			
		CDF		D0	
		Min.	Max.	Min.	Max.
Reconstruction for DC at 70% efficiency		48,000	72,000	58,000	143,000
Monte Carlo		10,000	40,000	20,000	40,000
Analysis		-		-	
Reprocessing		24,000	24,000	30,000	70,000
Stripping		10,000	10,000	-	
Total (processing types overlaps considered)	**by year 2000**	50,000	70,000	80,000	160,000
		130,000 - 230,000			
	by year 2001	72,000	96,000	110,000	253,000
		182,000 - 349,000			

The number of CPUs and Cost may be estimated by making the following assumptions:

- Measurements show that a 200 MHz PC can provide 115 MIPs. Extrapolating this means that a 400 MHz PC can deliver 230 MIPs and that a 500 MHz PC can deliver 287 MIPs.
- A dual processor 400 MHz PC with 6 GBytes of disk will cost approximately $3200.
- No extra disk will be needed for caching, data files are no more than 1 GByte and at most 2 data files will be needed per processor. Thus, in this estimate, a data disk in addition to the system disk is assumed to be unnecessary.
- A switch will cost approximated $50,000.

Using the total MIPs considering overlaps in processing types give the scope of the project as:

Estimated MIPs			Number of Dual Processor PCs		Cost in $K (if bought in 1998)
			400 MHz	500 MHz	
Total year 2000	CDF	50,000 - 70,000	110 - 150	90 - 125	
	D0	80,000 - 160,000	175 - 350	140 - 280	
	CDF+D0	130,000 - 230,000	285 - 500	230 - 405	962 - 1,650
Total year 2001	CDF	72,000 - 96,000	160 - 210	125 - 170	
	D0	110,000 - 253,000	240 - 550	195 - 440	
	CDF+D0	182,000 - 349,000	400 - 760	320 - 610	1,330 - 2,482

These estimates will be refined by CDF and D0 over time.

2.2 OVERALL ARCHITECTURAL DECISIONS

Using PC technologies in the Farms is desirable because:

- of their low cost
- Linux offers more standardization for administrators
- experience at Fermi with PCs to date has been very favorable

However, other types are not ruled out (e.g. SGI, AIX, DEC workstations or SMPs), particularly for the role of file servers or centralized control systems.

In previous runs, the Farms have relied heavily on locally attached tape drives for I/O. While cost effective in the past, their are overall a high maintenance item. With the advent of the Centralized Mass Storage System, it will be more efficient and cost effective to eliminate locally attached tape drives to farm hosts and go through Mass Storage System for data instead.

To simplify CDF and D0 reconstruction software, files based control software is preferred over event based software. Thus, Cooperative Processes Software (CPS) will not be used and will no longer be supported beyond Fixed Target 1999.

Minimize the number of unique batch architectures supported by the Fermi Computing Division. LSF, a commercial batch system developed by Platform Computing, has been used successfully at Fermi for several years and my prove beneficial as the core of the Farms Batch System.

The hardware configuration of hosts should consider two types:

- a few with adequate disk, good network access, and act as NFS servers
- many with limited disk, slower network access and act as NFS clients

Hosts should be logically classified into two major sets:

- Job Manager hosts that allow full user access, job submission & central control point for user jobs
- Execution hosts that are dedicated to computational and/or I/O tasks and allow relatively restricted interactive user access. Execution hosts may be further divided into two subtypes:
  - — I/O hosts, suited for I/O-bound batch tasks with relatively large disk space and fast network access to mass storage and/or site-wide network and
  - — CPU (worker), hosts with limited disk space, dedicated primarily to CPU-bound tasks

The Farms Batch System architecture itself should not require such division of execution hosts into types and should be flexible enough to allow different ways of logical division of hosts into types. Actual configuration of execution hosts will be dictated by hardware capabilities and volume and nature of computations performed by end users. The Farms Batch System should allows fast, flexible and non-intrusive fine tuning of its configuration.

3.0 HARDWARE AND NETWORKING ARCHITECTURE

3.1 RUN II HARDWARE MODEL

A Typical Farm will include:

- One or more Job Manager (JM) hosts. JM hosts will run LSF and allow interactive user sessions. JM hosts will run approximately 100 - 1000 Job Manager processes depending on users demand and Farm Batch System configuration. (See “Job Manager (JM)” on page 14 of the “Software Architecture” section). JM processes are mostly dormant processes, and will not consume significant amount of system resources, except for virtual memory. JM hosts will have network connection to all hosts in the cluster. This network connection will be used for communication between JMs and other components of the Farms Batch System.
- Up to 500 Execution hosts. User processes will run on Execution hosts. The number of Execution hosts in the cluster will be reversely proportional to performance of individual computer. Execution hosts could be single or multiprocessor machines.

There are two possible architectures for the Execution hosts:

1) Differentiate each Execution Host as either a CPU host (or Worker host) or an I/O host. From the Farms Batch System standpoint, these two types of hosts will be different only by type name and maximum number of processes to run in the same time.
   - I/O hosts will be used for I/O-bound user processes and, in the same time, serve as disk servers for CPU hosts, hosting Farms Data Storage (See “Farms Data Storage (FDS)” on page 15 of the “Software Architecture” section). FDS will control scratch disk space used for transferring data to and from the Central Mass Storage System (MSS) and the outside world. I/O hosts will have fast network connection to MSS. Note that since a JM process consumes very little system resources (CPU time, network bandwidth), it may run on I/O host, therefore some I/O hosts may be JM hosts in the same time. Approximately 10% of all Execution hosts will be I/O hosts.
   - CPU-bound user processes will be running on CPU hosts and approximately 90% of all Execution hosts will be CPU hosts. Depending on the CPU host’s performance characteristics, it will be configured to run one or more user processes in the same time. Also, CPU hosts will have network connection to each other to enable IPC between cooperating user processes. There are several possible ways in which user processes will be able to access data stored on I/O hosts:
     A) “directly” through NFS
     B) using FIO/RFIO package (an I/O library that allows to read/write via TCP/IP)
     C) downloading data to local disk and uploading data back via rcp or other tools

     In cases A) and B) CPU hosts have only small amount of local disk space to accommodate I/O needs of CPU-bound processes running on the host. The majority of cluster disk space resources are physically attached to some number of I/O hosts. Each I/O host works as a NFS server for CPU hosts and all other I/O hosts. In this case FDS can be used to manage centralized disk space resources. In case C) each CPU host must have enough local disk space to accommodate input and output files for maximum number of processes allowed to run in the same time on the host. Current estimate of disk space needed is 4 GBytes per process. Configuration C) with the local disk on worker hosts will have better I/O

performance then A), will not required additional programming that will be necessary in configuration B), but will increase the complexity of over all job control, additional cost of disk and will require additional effort from administrators or users to manage local disk space.

There are several advantages to this approach:

— Easily optimize access to MSS
— Relatively few fast network connections to MSS
— Optimal CPU utilization
— Centralized disk space management is possible

The only apparent disadvantage is that additional data transfer between I/O and CPU host is required.

This architecture is shown in FIGURE 1.

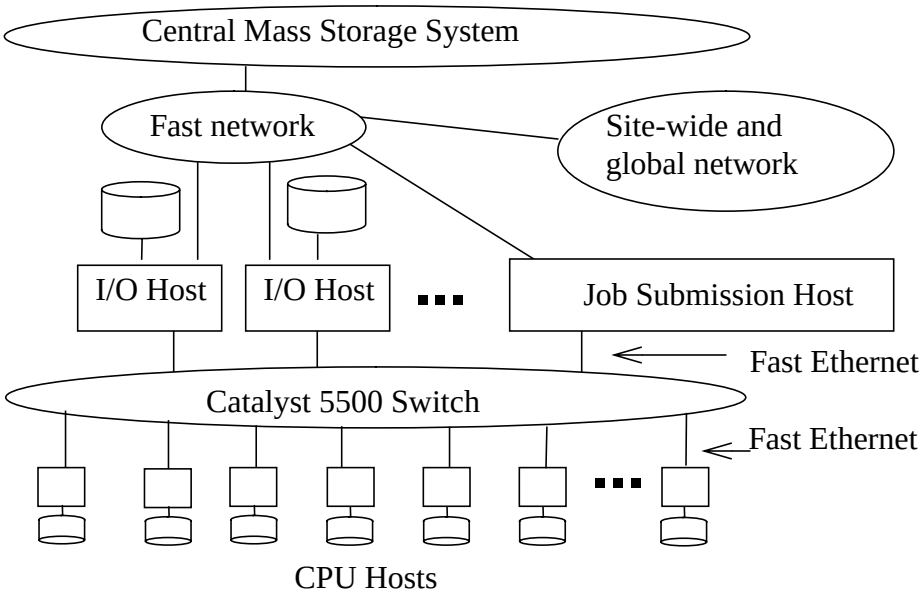


FIGURE 1. Execution hosts differentiated as I/O versus CPU hosts

2) No differentiation between I/O and CPU hosts. In this architecture, all disk space is distributed across hosts and is used not only for storing input and output data of running processes, but also for long-term data storage and (pre)staging. Each host will manage its own disk space and will access MSS on its own. FDS in this case seems to be unnecessary. Current estimate of disk space needed per host is the sum of:

— 2 GBytes for each input file being read by active analysis jobs
— 2 Gbytes for each output file being written by active analysis jobs
— 2 Gbytes for each input file being written by input staging jobs
— 2 Gbytes for each input file being written by output staging jobs.

Thus, 12 Gbytes would be required on a 2 processor host with one input staging jobs, one output staging job and 2 analysis jobs.

This configuration eliminates extra data transfer within the cluster and the dependency of CPU hosts on I/O hosts, but it has several essential drawbacks:

— Inefficient CPU utilization
— Drastic increase in number of network connections to MSS
— Users have to provide the mechanism to manage local disk space
— Additional effort will be required to optimize access to MSS
— Problem identification and system monitoring become more difficult

This architecture is shown in FIGURE 2.

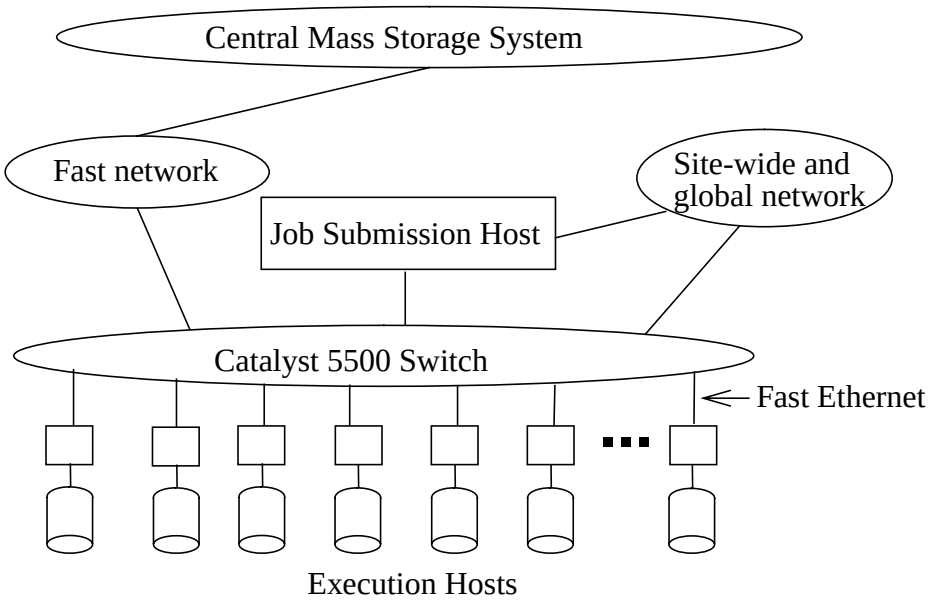


FIGURE 2. Execution hosts with no differentiation as I/O versus CPU hosts

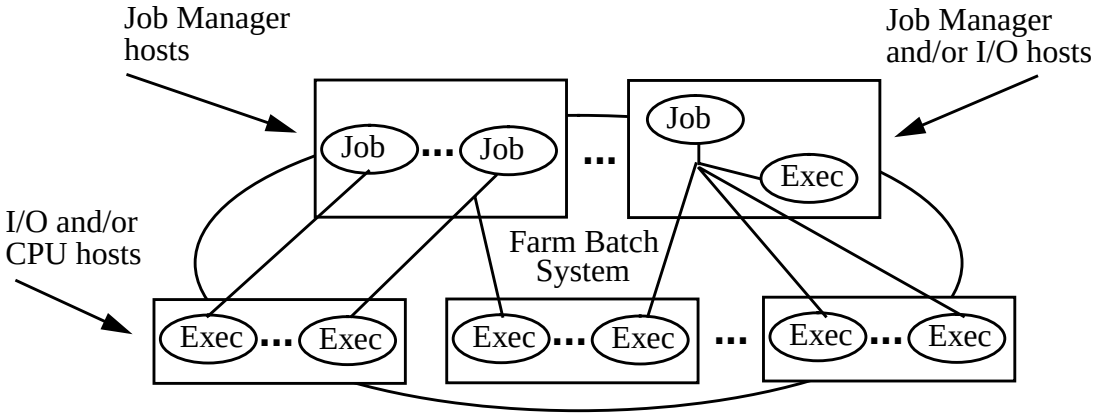
4.0 SOFTWARE ARCHITECTURE

4.1 OVERVIEW

The major software element of the Run II Production Management software is a batch system that works in concert with the Run II CDF and D0 Data Access Methods. The Farm Batch System will provide:

- Job Control for software
  - — queue users job
  - — dispatch user process(es) for execution to Execution hosts
  - — report job/process status to user
  - — coordinate job steps (sections) execution
  - — notify user when execution steps are complete
- Resource Management
  - — host status
  - — buffer/scratch disk status
  - — authorized users
  - — number of allowable executables per host
- Management Tools
  - — job submission/cancellation
  - — job hold/release
  - — job monitoring
  - — job history & statistics
  - — farm shutdown/start-up

Assuming the above, FIGURE 3. graphically illustrates this:



Job: A user job which specifies one or more CPU intensive tasks and/or I/O tasks to be remotely executed.

Exec: A CPU intensive executable or an I/O intensive executable

FIGURE 3. The Farm Batch System in a generic production context

4.2 SOFTWARE COMPONENTS

The Farm Batch System (See FIGURE 4.) will be composed of the following software elements:

- LSF for high level batch control on Job Manager hosts including ELIM interface specification for specific Farm load indices. (For convenience, non-farm production jobs may use the same LSF batch cluster, though such jobs will not have access to Execution hosts.)
- Farm Load Information Manager Daemon (FLIMD) for central management of all execution hosts
- Job Manager (JM) to control and monitor a particular section of Farm job on a given Job Manager host
- Farm Daemon (FARMD) to start, monitor and control user processes on a particular execution host
- Farms Data Storage (FDS)- a scratch disk space manager to be used as a data buffer between the system, mass storage and the rest of the world
- API for job submission, monitoring, status, history, administration, etc...
- Configuration file for farm configuration
- Farms Logging Daemon (FLOGD) to log all error and output messages in a single directory

Farm production jobs will be described by a Job Description File (JDF), which will consist of one or more Job Sections. A job will be submitted with the `farms submit` command. This command will invoke a LSF `bsub` command for each Job Section and specify the JM as the executable. The only argument to the `farms submit` command is the file name of the Job Description File (JDF). See “Job Description File (JDF)” on page 12 for more information on JDF and Job Sections.

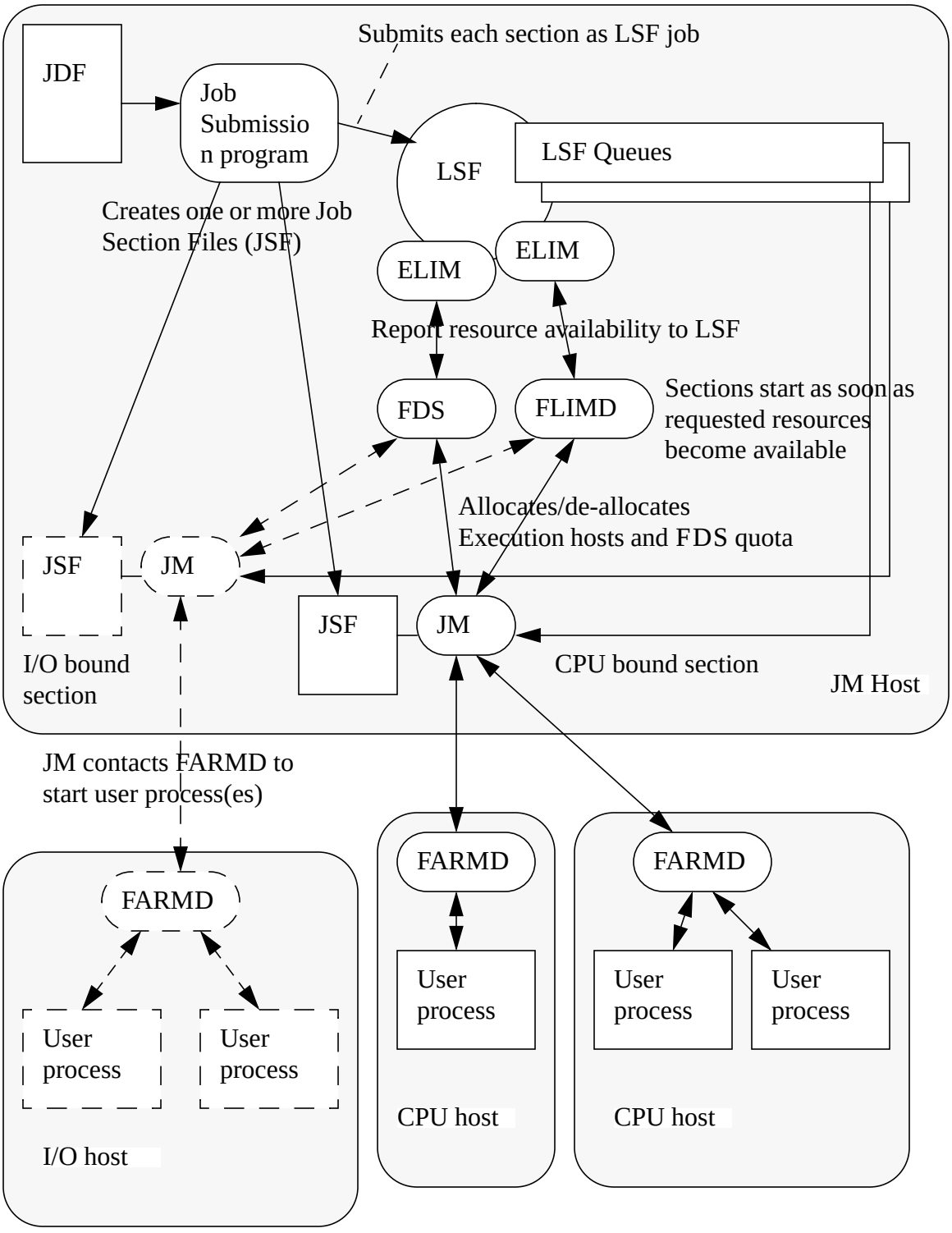


FIGURE 4. Farm Batch System Components

4.2.1 LSF

LSF is a commercial batch package. It will provide the following:

- Job submission and monitoring
- Job completion status reporting
- Job log file handling and delivery back to user
- Job cancellation
- Job queue management

LSF will use an ELIM (See Section 4.2.3 on page 13) as the interface to the FLIMD (Section 4.2.4 on page 14) to gather execution host and FDS disk space availability information. Based on this information and job queue scheduling parameters, LSF will start Job Manager processes (See Section 4.2.5 on page 14). LSF is used as a mechanism to start JM processes, but does not directly start user processes. User processes running on the farm are completely out of the control of the LSF system. Users do not need to be aware that LSF is being used. A set of API commands will be supplied that completely hides LSF from the user.

A queue level pre-exec command will run on each queue that will define environment variables that will provide the JM with the following:

- Host type
- CPU time limit
- Real time limit
- User/group permissions to use the queue

4.2.2 Job Description File (JDF)

This file contains all information needed by the Farm Batch System to submit a user's job. The path of the file is supplied to the farms submit command. The JDF file consists of one or more Job Sections. Each Job Section has a unique label. Users will be able to specify dependencies between sections (see the example below). LSF will be used to implement inter-section dependencies. Each Job Section has the following types of information:

- LSF queue to submit the job to
- User program pathname and arguments
- Number of processes to run in parallel.(1 by default)
- A flag indicating whether the job and all user processes should be cancelled if one user process dies. If the flag is set, then the death of one user process will not cause the entire set of user processes to be cancelled. The default action is to cancel all user processes for a section if one dies.
- FDS quota to reserve for this section (0 by default)
- Dependencies on other sections (none by default)
- What directory to put the stdout and stderr of each user process

A Job Section is an atomic element of a job. Each section is executed on one or more hosts of the same type. Each process of the section executes the same executable (most likely, UNIX shell script) and is provided with the same set of user parameters. Using dependencies between sections, user may establish (primitive) synchronization between sections. Section completion

code, determined by exit codes of individual processes, maybe used to specify conditions which are necessary to start next section.

A job submission program (see below) will process and validate JDF, create a Job Section File for each section and submit one LSF job per section.

An example of JDF (note that actual syntax may change) is:

```
SECTION input                          # section label
    EXEC = /user/dump -vsn XY123       # program and arguments
    QUOTA = 8G                         # FDS quota to reserve
    QUEUE = IO                         # queue to submit to
                                       # no dependencies for this section
SECTION processing                     # second section
    QUEUE = CPU_120min                 # queue to submit to
    QUOTA = 4G                         # FDS quota to reserve
    N =2                               # 2 processes to run
    NEED = 0                           # If one dies - proceed
    EXEC = /usr/user/myjob -vsn XY123  # program and arguments
    DEPEND = done(input)               # Start this section when input is done
    STDOUT = /usr/home/job_out         # Where to put processes stdout.
    STDERR = /usr/home/job_err         # Put stderr in job_err
```

In this example, section “input” runs on an I/O host and dumps a tape with VSN XY123 to disk. It requests 8 GByte of FDS quota for that. Let’s assume that it produces 2 files and puts them to well known location identified by VSN (e.g. /scratch/XY123/file1 and file2). Section “processing” will not start until

- section “input” finished successfully due to DEPEND clause (done(input)) and
- there are at least 2 free CPU hosts in the cluster
- there is at least 4 GByte FDS quota available for each of them

When section “processing” starts, two user processes start on Execution hosts in the same time and each of them processes one of files produced by previous section. Each of those two processes will be given the number of the process within the job. One process will get number 1 and the other - number 2. This way each of them will know which file (file1 or file2) to process. Similarly, stdout and stderr files will be uniquely named within the specified directory.

Job submission may be done on any Job Manager host. There is nothing that precludes that LSF on such hosts cannot be used for non-farm jobs. For example, users may be able to submit standard LSF jobs for daily user area maintenance. However, such jobs will not have access to execution hosts and will run on JM hosts.

4.2.3 External Load Information Manager (ELIM)

From LSF standpoint, an ELIM is supplied by user as a module which monitors user-defined load parameters. For the Farms Batch System, there is an ELIM process to contact the FLIMD requesting the number of available hosts in the Farm Batch System, and from FDS, the ELIM receives information on disk space.

4.2.4 Farms Load Information Manager Daemon (FLIMD)

The FLIMD process monitors Execution host availability and status information making it available to LSF (through the ELIM), to users and to cluster administrators. Host status is maintained in real-time. The FLIMD configuration file will have one entry per execution host (see Section 4.4.1 on page 17)

The type of information kept in memory will be:

- list of LSF job ids for JMs which have user processes running on the host
- time of last seen heart-beat message from each FARMD
- host status (up or down)

Based on Execution host availability information, LSF will start JM processes, when requested resources (given number of Execution hosts of given type) become available. When it starts, the JM will contact the FLIMD and allocate 1 or more hosts of certain type. When user job finishes, the JM will de-allocate the resources. If the JM terminates before releasing the resources, the FLIMD will detect that and deallocate the resources automatically. The JM may detect that certain Execution hosts are is unavailable and notify the FLIMD. The FLIMD will listen to "heartbeat" messages from FARMD processes (See Section 4.2.6 on page 15) and use them to update host status. The FLIMD will read its configuration file, which will be used to specify maximum number of processes to run on each Execution host and host type. There will be one and only one FLIMD running per cluster at any time.

The FLIMD maintains open TCP stream connection to all running JMs. When the FLIMD starts, it waits for some time for all existing JMs to reestablish connections and to send updated information on Execution host usage. Information collected this way, heart beat messages from FARMD's and a static FLIMD configuration file are all sources of information FLIMD needs to start. This allows the Farms Batch System to be robust with respect to FLIMD failure.

4.2.5 Job Manager (JM)

A JM process will be started by LSF and will be run as a user process. There will be one JM per Job Section in a JDF file. The JM will receive from LSF all information that is necessary to start user processes (e.g. Job Id, user's program name and arguments, number of processes to run, etc.)

On start, the JM will contact the FLIMD to allocate given number of hosts of a certain type. After receiving list of allocated hosts, the JM will contact FARMD processes on specified hosts and pass process related information such as job name, user id, user's program name with arguments etc.

A JM process stays in existence until all user processes have finished successfully or one of the following has happened:

- The FLIMD could not allocate the necessary amounts of hosts
- user cancels the job
- A FARMD process died or host became unavailable
- one of user process exit with non-zero code

A JM will also perform the following actions:

- in case of lost connection with the FLIMD periodically tries to reestablish connection and

update the FLIMD with current information (list of hosts where user processes are running)
- notify the FLIMD in case when a FARMD on certain host is not available
- notify the FLIMD when an allocated host is not needed and could be released
- notify a FARMD in case of job cancellation
- produce a job log file

A JM will exit with zero exit code only in case when all user processes have been finished successfully, otherwise the exit code will be equal to 1.

4.2.6 Farm Daemon (FARMD)

The FARMD is essentially a Farms Batch System agent running on every Execution host. There will be one FARMD running per host. Responsibilities of the FARMD include:
- Accept connections from JMs
- Keep connection to each JM open until the JM disconnects or last user process finishes
- Start user processes as requested by the JM
- Pass the number of cooperative processes, the number of the process in the job (1,2,3, and so on) and the LSF job number in UNIX environment variables to user process
- Store user process' standard output and error files on disk
- Enforce CPU and day time limits
- Notify the JM when a user process exits
- Terminate user processes if requested by a JM
- Provide status and statistical information about user processes
- Periodically send heart-beat messages to the FLIMD to update the host status information

The FARMD will be running with root privileges.

4.2.7 Farms Data Storage (FDS)

FDS is the Farms Disk Space manager for scratch and staging disk space. It will be responsible for:
- Distribution of user data across scratch disk space. FDS will manage one or more UNIX file systems physically attached to one or more computers. Those computers may or may not be included in Batch System. FDS will place user data based on space availability and user (or group) disk space quota, if any;
- Reporting disk space availability to LSF to allow user jobs to specify their requirements for scratch disk space and wait in LSF queue until those are satisfied;
- FDS will be responsible for cleaning unused data from disk based on user-defined parameters such as time of last access to the file, file and/or project life time.

FDS will maintain virtual file system database, which will be used as a namespace, similar to the one provided by FMSS and ENSTORE. All scratch disk space will be logically divided between groups and/or individual users by quota mechanism. Further, parts of user or group disk space will be divided into "projects", which will correspond to Farms Batch System jobs or groups of jobs. User interface to FDS will include operations like:
- Uploading/downloading a file to/from FDS

- Reserving/unreserving disk space for future use by a project
- Querying disk utilization statistics
- Translation of virtual file name into physical one. This feature may be useful in cases when data is in fact stored on local or NFS accessible physical disk
- Also, it may include “direct” data access through FIO-like package

4.2.8 Farms LOGging Daemon (FLOGD)

The FLOGD is a daemon that receives, confirms, and logs messages from all components of Farm Batch System. UDP protocol will be used for the FLOGD communication to reduce consumption of system resources.

4.3 ERRORS AND RECOVERY

Careful consideration has been given to various scenarios of failure of each Farms Batch System component. The goal is to minimize impact of such failures on user jobs and overall system performance. A cron job will run periodically to parse the log file. If errors are found in the log file, the cron job will automatically notify appropriate personnel (e.g. via email)

4.3.1 Failure of LSF

If LSF terminates or has to be restarted, FLIMD, FDS, JMs, FARMDs and user processes will continue to run. LSF failure will result in:

- Users will not be able to submit new jobs
- Users will not be able to check current status of their jobs
- Users will not be able to cancel their jobs

When LSF restarts, it will learn about available resources and recover. No user jobs will be terminated because of LSF failure. Certain LSF administrative actions, such as closing an LSF host, can cause suspension of the JM process. The effect of this on the user processes (which are running outside of LSF’s control) has yet to be determined. One option is to suspend these processes as well.

4.3.2 Failure of ELIM

In case of ELIM failure, jobs will be held in LSF queues until the ELIM restarts (approximately 90 seconds.)

4.3.3 Failure of FLIMD

FLIMD/JM and FARMD/FLIMD communication protocols are designed so that the FLIMD, when it restarts after a failure, receives all necessary job status, Execution host status and availability information from JMs and FARMDs. For the time when the FLIMD is not running:

- New Job Manager processes will wait until FLIMD starts
- The ELIM will report that no hosts are available, preventing LSF from starting new jobs
- The JMs of jobs which are already running will try to reestablish connection to the FLIMD,

and when it comes back up, update the FLIMD with their current status

- If a job finishes before the FLIMD restarts, the JM exits.
- FARMDs continue sending heartbeat messages to the FLIMD. Connection-less UDP protocol will be used for this.

This algorithm allows FLIMD failure and recovery to have very little impact on users.

4.3.4 Failure of JM

If a JM terminates unexpectedly, all FARMDs it had a connection to gracefully terminate user processes by sending SIGUSR1 signal first, and after some (configurable) period of time (e.g. 10 minutes) send SIGKILL signal. The FLIMD considers all hosts allocated by the failed JM as available after the grace period of time elapsed since the JM disconnection. In other words, failure of a JM causes graceful job section termination and may cause dependent sections not to start.

4.3.5 Failure of FARMD

When a FARMD terminates unexpectedly, one or more JMs which have user processes running on corresponding execution host detect that immediately as network connections to FARMD break. These JMs notify the FLIMD that this Execution host is unavailable and, if user requested, may gracefully terminate all other user processes, or may allow them to continue execution. When the FARMD comes back up, it sends heartbeat message to the FLIMD, making itself available to run user processes.

4.3.6 Failure of FDS

In case of FDS failure, all processes in phase of downloading or uploading of data will fail. LSF will hold all jobs requesting some FDS disk space as resource. Processes which have downloaded their data onto local disk on a CPU host or those accessing data directly or through NFS will continue to run. Processes which have finished their computations and attempting to upload data back into FDS will block until FDS recovers.

4.3.7 Failure of FLOGD

If the FLOGD happens to terminate, each component of the Farms Batch System (e.g. FARMD, JMs, FLIMD, FDS) would hold up to 10 messages in their perspective queues. When the FLOGD has been restarted, these messages would be sent to the FLOGD at that time to be logged.

4.4 LOCAL DATABASES

4.4.1 Configuration Files

Each component of Farms Batch System reads from a farm configuration file. This file will be stored on a disk which is NFS mounted at all the Farm Batch System hosts. Location of configuration file will be defined in FUE UPS setup scripts by an environment variable.

The FLIMD has its own configuration file which defines the cluster configuration. This file will have the following type of information about each Execution host:

- Host name;
- Host types it belongs to. A host may belong to more than one type;
- Maximum number of user processes to run on the host per type (e.g. 2 WorkerTypeA and 2 WorkerTypeB on the same host)

4.5 DESIGN ISSUES

Most software will be written in a high level scripting language such as PYTHON.

4.6 USER AND ADMINISTRATOR API

The API will provide both a command line and GUI interface. All functionality will be implemented in the command line interface. The API consists of the following types of commands:

- Job Submission Program (SUBMIT)
- Job/process status monitoring tools (STATUS)
- Job control tools (KILL)
- System monitoring tools (QUEUES,HOSTS)
- Job history, viewing log files.

All command are issued using the farms command:

```
farms [API Command] [args]
```

The following subsections provide examples to show the proposed functionality of several of these commands.

4.6.1 Job Submission

Job submission is done using the `farms submit` command. The user supplies a JDF file as an argument to submit the job. The following example submits a job whose JDF file is located in the users home area under a file called myjob.jdf:

```
> farms submit ~/myjob.jdf
Farm Job < 4548 > has been submitted ...
```

This will submit an LSF job that dispatches a JM process. The JDF file contains information that will be used to determine what executable to run, what type of hosts to use, how many parallel processes to run, etc...

After the job has been submitted, a farm job id will be printed to stdout. This job id can be used to monitor or cancel a farm job.

4.6.2 Job Monitoring

Farm jobs can be monitored by using the `farms status` command. The command can look at jobs for a particular user, and individual job, or all farm jobs running on the system. The following

example shows all jobs in the batch system by user fromm:

```
> farms status -u fromm
Farm Jobid:  4548
-------------------------------------------------
 Step Name:  one
 Host:  fnpc24.fnal.gov

 Process Number:  2
 PID   CPU   ACPU  CMD
 2658  0     0     sh /home/fromm/sleep_well.exe
 2659  0     0     sleep 1000

 Process Number:  1
 PID   CPU   ACPU  CMD
 2656  0     0     sh /home/fromm/sleep_well.exe
 2657  0     0     sleep 1000
-------------------------------------------------
 Step Name:  two
 Pending:   Job dependency condition not satisfied;

-------------------------------------------------
 Step Name:  three
 Pending:   Job dependency condition not satisfied;
```

4.6.3 Job control tools (KILL)

Farm jobs can be cancelled using the farms kill command. This will kill all processes that are running under a particular farm id. The following command kills the job that was submitted in the previous section:

```
> farms kill 4548
Farm job < 4548 > being killed...
```

4.6.4 System monitoring tools (QUEUES,HOSTS)

These commands will supply information on the currently configured hosts and queues in the system.

5.0 ADMINISTRATION AND SUPPORT ISSUES

The following items must be installed to run the Farm Batch System.

- Python (or other scripting language if used)
- LSF on all JM hosts
- Products area NFS mounted to all Execution hosts.
- On each Exection host, a FARMD scratch area directory must be created. This directory can be found in $FARMS_ROOT/config/farms.cfg.
- The FLIMD configuration file must be accessible on the the FLIMD host. The location is determined when the FLIMD is started.
- On the FLOGD host, a log directory must be created. This directory can be found in $FARMS_ROOT/config/farms.cfg.